SDFuzz: Target States Driven Directed Fuzzing

Penghui Li[1,2], Wei Meng[1], Chao Zhang[2,3]

[1] The Chinese University of Hong Kong

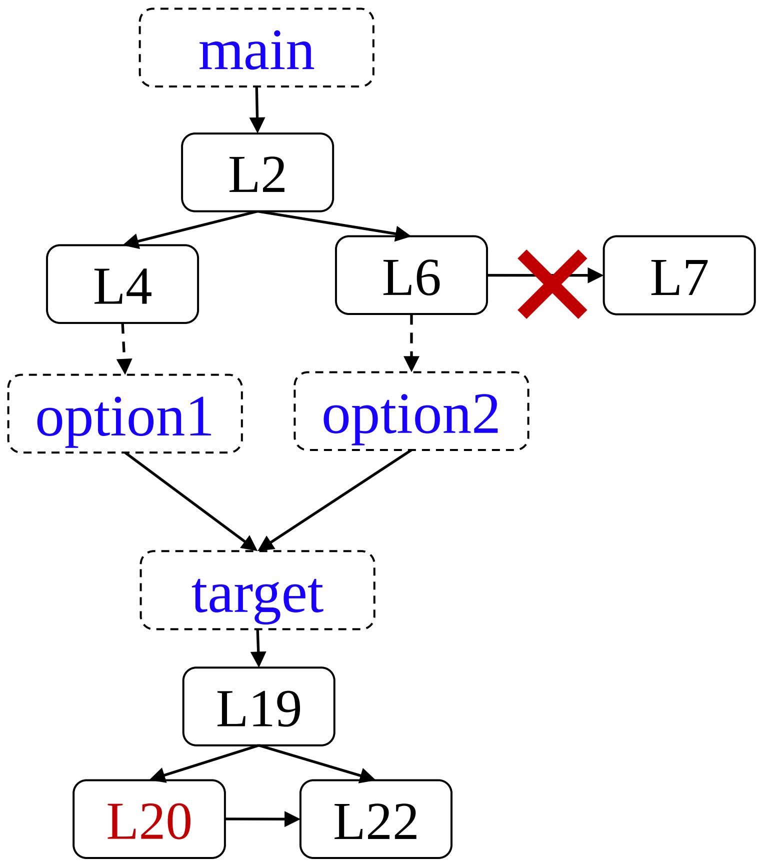
[2] Zhongguancun Laboratory [3] Tsinghua University

Directed Fuzzing

- Directed grey-box fuzzing (DGF) tests towards highly valuable locations
  - PoC generation
  - Vulnerability validation

- Recent works prunes irrelevant code paths

Static analysis of ICFG can hardly decide which relevant code path is better

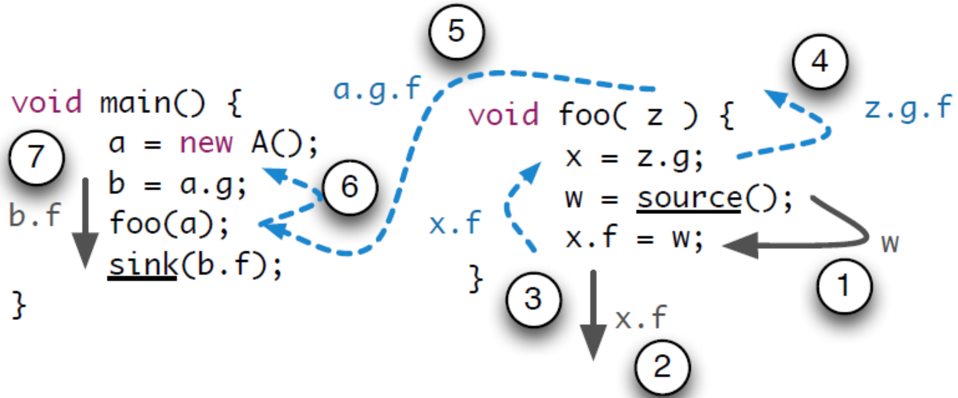


Target States

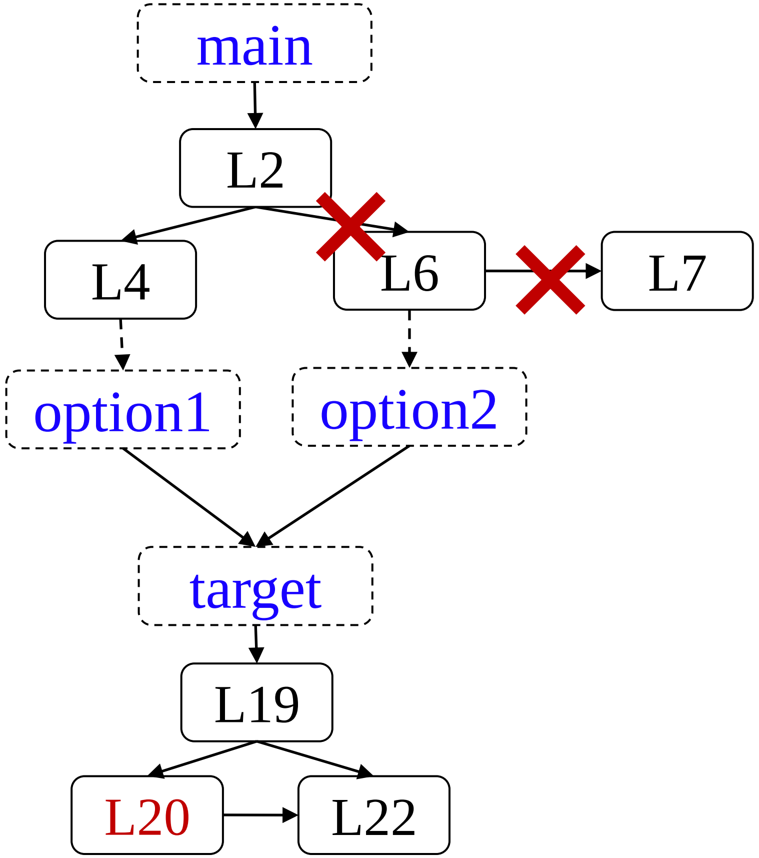
- Major tasks of DGF provide detailed descriptions of the target vulnerabilities

```
0x... in target       file.c:20
0x... in option1      file.c:15
0x... in main         file.c:4
0x... in in __libc_start_main
```

Crash dump



Taint flow

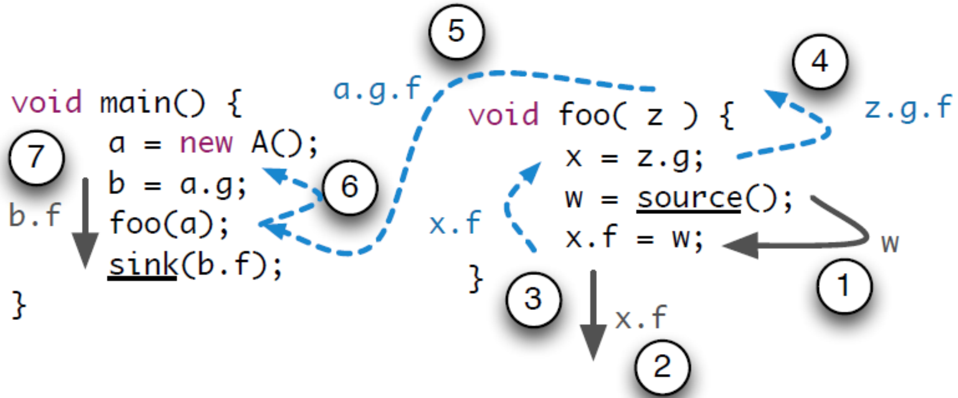


Target States

- Major tasks of DGF provide detailed descriptions of the target vulnerabilities

```
0x... in target        file.c:20
0x... in option1       file.c:15
0x... in main          file.c:4
0x... in in __libc_start_main
```

Crash dump



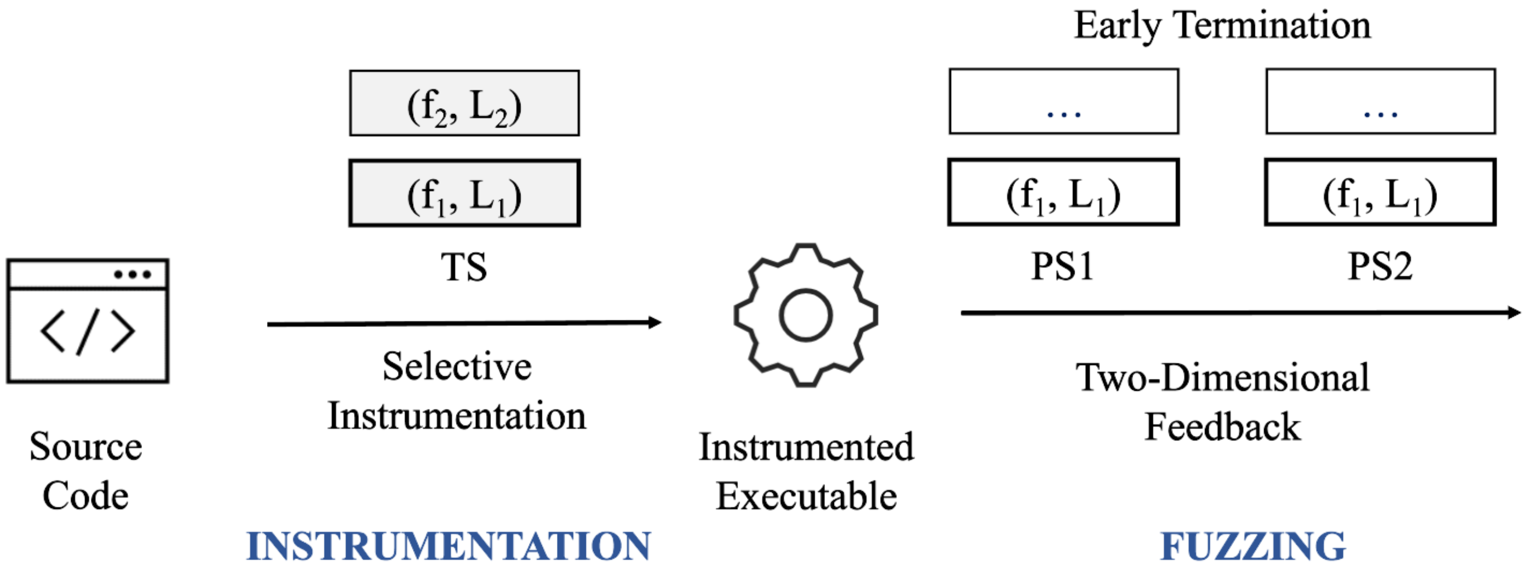
Taint flow

Formalize a target state as a sequence of ordered function calls and their invocation contexts

$$[(Func1, Ctx1), (Func2, Ctx2), \dots, (Funcn, Ctxn)]$$

SDFuzz in a Nutshell

- Reduced testing scope
  - Test only required code specified in target states
- Improved throughput
  - Early terminate executions that cannot reach target states



Required Code Identification

- Identify code required to reach target states
  - A subset of the code for reaching target sites
- Functions specified in target states and their dependent functions
- Coverage feedback from only required code
  - Selective instrumentation
  - Other code is hidden from the fuzzer

Algorithm 1: Required code identification.

```
input  : TSs, ICFG
output : requiredFuncs
initRequiredFuncs ← [ ]
requiredFuncs ← [ ]
for TS ∈ TSs do
    for f ∈ TS do
        initRequiredFuncs.insert(f)
        funcs ← backwardAnalysis(f, ICFG) // get
          functions with intra-procedural dependencies
        initRequiredFuncs.insert(funcs)
    end
end
while ! initRequiredFuncs.empty() do
    f ← initRequiredFuncs.remove()
    if f ∉ requiredFuncs then
        requiredFuncs.insert(f)
        callees ← getCallees(f, ICFG) // get callees
          of f
        initRequiredFuncs.insert(callees)
    end
end
return requiredFuncs
```

Early Execution Termination

- Early aborts the executions that cannot reach the target states
  - Runtime program state monitoring
  - Check the deviation of current program state against target state

```
1  void main() {
2      int x = input();
3      if(x < 10)
4          option1(x); //(1)
5      else
6          option2(x); //(2)
7      clean();
8  }
9
10 void option2(int opcode) {
11     target(opcode);
12 }
13 void option1(int opcode) {
14     check();
15     target(opcode + 5);
16 }
17
18 void target(int arg) {
19     if (arg <= 20) {
20         assert(arg < 5);
21     }
22     ...
23 }
```

TS1	PS1	PS2	PS3
(target, L15)			
(option1, L4)	(input, L2)	(option1, L4)	(clean, L7)
(main, libc)	(main, libc)	(main, libc)	(main, libc)

PS1: deviation (input, L2) could be recovered into **(option1, L4)** because there is a path from L2 to L4

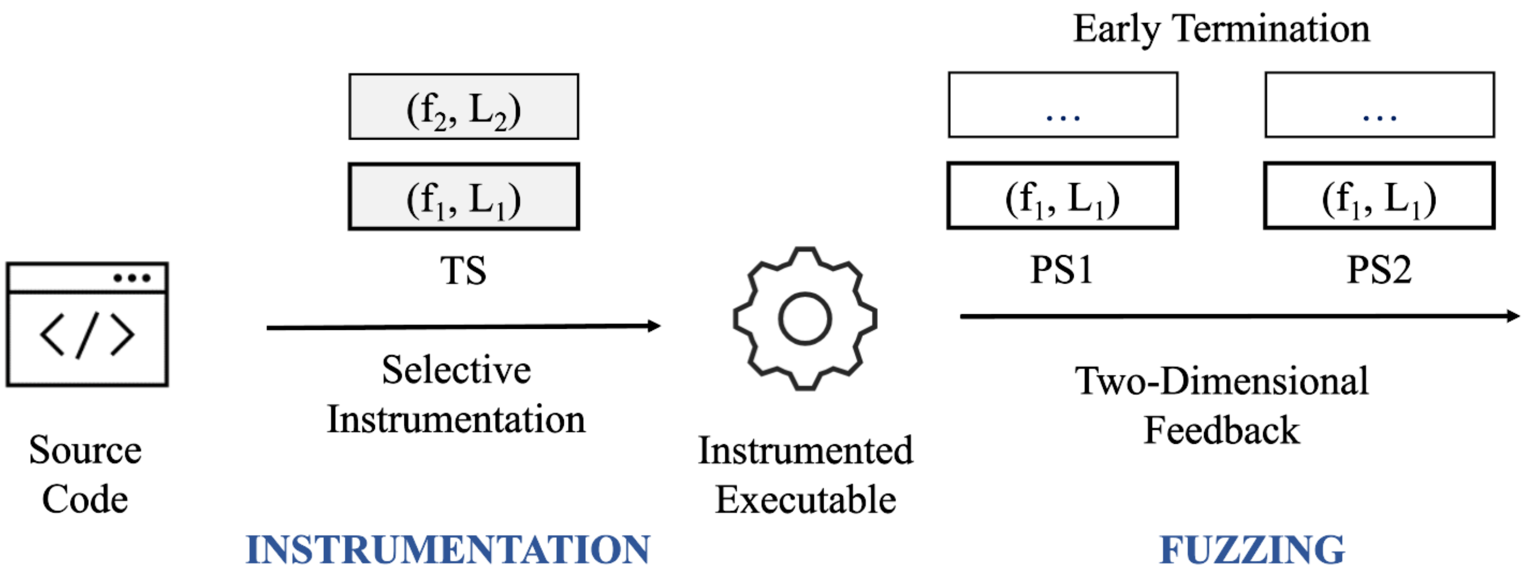
PS2: no deviation

PS3: deviation (clean, L7) could not be recovered because there is no path from L7 to L4

Feedback Mechanisms

- Target state feedback
  - Calculate how close current program state is to target states
- Improved distance feedback
  - Prior solutions consider every edge in CG equally
  - Design a precise edge weight

$$weight\left(f_i, f_j\right) = min\left(d_{f_i}\left(BB_{f_i-start}, BB_{f_j-call-site}\right)\right)$$



Evaluation

- What is the capability of SDFuzz in exposing vulnerabilities?
- How do the techniques contribute to SDFuzz’s performance?
- How effective is SDFuzz in discovering new vulnerabilities?

Vulnerability Exposure

- 45 vulnerabilities selected from Google Fuzzer Test Suite, AFLGo, etc.
- AFLGo/WindRanger/Beacon/SieveFuzz/SDFuzz exposed 36/37/34/40/44 cases
- SDFuzz used shortest time in 77.8% of cases

ID	Program	Location	AFLGo			WindRanger			Beacon			SieveFuzz			SDFuzz
			Time	Factor	p-val	Time	Factor	p-val	Time	Factor	p-val	Time	Factor	p-val	Time
1	libming	decompile.c:349	216	2.45	0.003	195	2.22	0.002	147	1.67	0.001	199	2.26	0.007	88
2	libming	decompile.c:398	268	1.71	0.008	348	2.22	0.003	194	1.24	0.050	282	1.80	0.030	157
3	LMS	service.c:227	5	1.67	0.009	8	2.67	0.006	3	1.00	0.001	3	1.00	0.001	3
4	mjs	mjs.c:13732	272	1.36	0.132	204	1.02	0.012	128	0.64	0.003	228	1.14	0.023	200
5	mjs	mjs.c:4908	8	2.67	0.007	5	1.67	0.004	5	1.67	0.006	3	1.00	0.001	3
6	tcpdump	print-ppp.c:729	608	4.68	0.004	708	5.45	0.003	CE	-	-	512	3.94	0.003	130
7	lrzip	stream.c:1747	372	18.60	0.005	251	12.55	0.003	38	1.90	0.001	176	8.80	0.003	20
8	lrzip	stream.c:1756	329	7.48	0.002	224	5.09	0.001	158	3.59	0.003	137	3.11	0.009	44
9	objdump	objdump.c:10875	785	5.38	0.002	752	5.15	0.008	235	1.61	0.003	327	2.24	0.003	146
10	objdump	dwarf2.c:3176	TO	-	-	618	7.92	0.001	CE	-	-	154	1.97	0.019	78
11	libssh	messages.c:1001	TO	-	-	TO	-	-	TO	-	-	TO	-	-	1,112
12	libxml2	valid.c:952	151	2.44	0.009	42	0.68	0.004	52	0.84	0.003	70	1.13	0.001	62
13	libxml2	messages.c:1001	217	1.43	0.003	209	1.38	0.002	78	0.51	0.003	192	1.26	0.018	152
14	libxml2	parser.c:10666	134	3.35	0.012	211	5.28	0.007	TO	-	-	78	1.95	0.009	40

Characterization

- SDFuzz eliminated 43.29% more unrequired code than SieveFuzz
- SDFuzz improved fuzzing throughput by 9.32 times compared to AFLGo

ID	Program	Location	AFLGo			WindRanger			Beacon			SieveFuzz			SDFUZZ
			Time	Factor	p-val	Time	Factor	p-val	Time	Factor	p-val	Time	Factor	p-val	Time
1	libming	decompile.c:349	216	2.45	0.003	195	2.22	0.002	147	1.67	0.001	199	2.26	0.007	88
2	libming	decompile.c:398	268	1.71	0.008	348	2.22	0.003	194	1.24	0.050	282	1.80	0.030	157
3	LMS	service.c:227	5	1.67	0.009	8	2.67	0.006	3	1.00	0.001	3	1.00	0.001	3
4	mjs	mjs.c:13732	272	1.36	0.132	204	1.02	0.012	128	0.64	0.003	228	1.14	0.023	200
5	mjs	mjs.c:4908	8	2.67	0.007	5	1.67	0.004	5	1.67	0.006	3	1.00	0.001	3
6	tcpdump	print-ppp.c:729	608	4.68	0.004	708	5.45	0.003	CE	-	-	512	3.94	0.003	130
7	lrzip	stream.c:1747	372	18.60	0.005	251	12.55	0.003	38	1.90	0.001	176	8.80	0.003	20
8	lrzip	stream.c:1756	329	7.48	0.002	224	5.09	0.001	158	3.59	0.003	137	3.11	0.009	44
9	objdump	objdump.c:10875	785	5.38	0.002	752	5.15	0.008	235	1.61	0.003	327	2.24	0.003	146
10	objdump	dwarf2.c:3176	TO	-	-	618	7.92	0.001	CE	-	-	154	1.97	0.019	78
11	libssh	messages.c:1001	TO	-	-	TO	-	-	TO	-	-	TO	-	-	1,112
12	libxml2	valid.c:952	151	2.44	0.009	42	0.68	0.004	52	0.84	0.003	70	1.13	0.001	62
13	libxml2	messages.c:1001	217	1.43	0.003	209	1.38	0.002	78	0.51	0.003	192	1.26	0.018	152
14	libxml2	parser.c:10666	134	3.35	0.012	211	5.28	0.007	TO	-	-	78	1.95	0.009	40

Component-Wise Analysis

- $AFLGo_{+et}$ early terminated 56.23% of the executions
- $AFLGo_{+df}$ achieved less significant improvement than $AFLGo_{+sf}$

ID	$AFLGo_{+si}$	$AFLGo_{+et}$	$AFLGo_{+sf}$	$AFLGo_{+df}$	SDFuzz
1	1.16	1.95	1.27	1.08	2.45
2	1.00	2.11	2.11	1.34	1.71
3	1.25	1.67	1.00	1.00	1.67
4	1.30	1.17	1.17	1.08	1.36
5	2.21	2.00	0.80	1.60	2.67
6	1.24	3.22	1.12	1.16	4.68
7	2.10	9.79	3.19	2.49	18.60
8	2.19	3.58	1.39	1.36	7.48
9	2.02	2.42	1.54	1.32	5.38
10	✓	✓	TO	TO	✓
11	TO	TO	TO	TO	✓
12	1.96	1.94	1.26	1.16	2.44
13	1.09	1.15	1.09	1.14	1.43
14	1.72	1.97	1.12	1.54	3.35
15	TO	TO	TO	TO	✓
16	1.28	1.49	1.07	1.06	2.98
17	1.38	4.02	1.94	1.13	4.26
18	1.50	1.50	1.39	1.69	3.00

Vulnerability exposure time against AFLGo

Each enables a feature atop AFLGo

- $AFLGo_{+si}$ for selective instrumentation
- $AFLGo_{+et}$ for execution termination
- $AFLGo_{+sf}$ for target state feedback
- $AFLGo_{+df}$ for distance feedback

New Vulnerability Discovery

- Integrate SDFuzz with saber checker of SVF into a fully automated solution
  - Static analysis had many false positives
- Vulnerability-triggering paths are exact paths reported by SVF for those validated vulnerabilities

Program	Statically Reported	SDFuzz Validated
libjpeg	46	2
tinyexr	22	1
pugixml	59	1
ffmpeg	32	0
Total	159	4

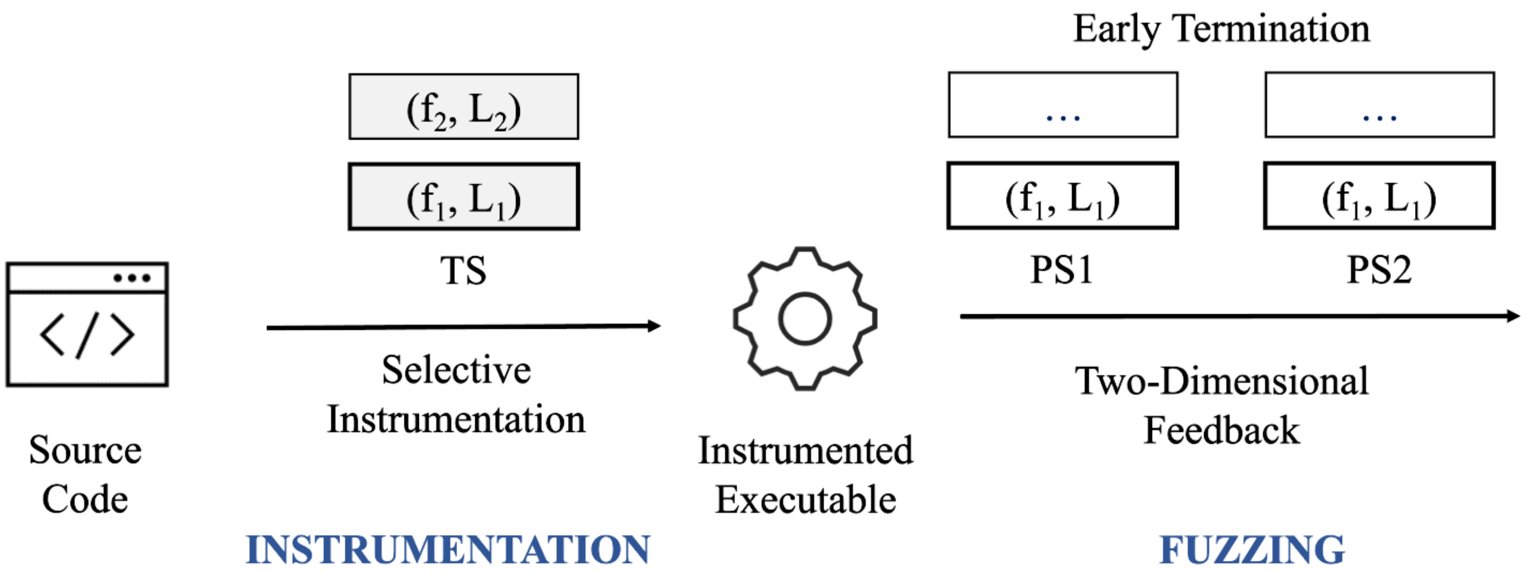
Discussion

- Requirement of target states
  - They might not be available in some scenarios like patch testing

- Overlook other valuable paths not included in target states
  - Reasonable trade-off
  - Infeasible ones dominate program paths
  - Paths stated in target states are preferred working ones

Summary

- Target states extracted from DGF tasks are helpful
- Eliminating unnecessary exploration greatly improve fuzzing throughput
- SDFuzz could effectively expose vulnerabilities and validate static analysis alerts



Thank You!

Feel free to contact me for follow-up discussions:

lipenghui315@gmail.com